(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) Publication number: **0 605 872 A1**

(12) EUROPEAN PATENT APPLICATION

(21) Application number: 93120940.7

(51) Int. Cl.5: G06F 9/38

(22) Date of filing: 27.12.93

(30) Priority: 08.01.93 US 2445

(43) Date of publication of application:
13.07.94 Bulletin 94/28

(84) Designated Contracting States:
DE FR GB

(71) Applicant: INTERNATIONAL BUSINESS MACHINES CORPORATION
Old Orchard Road
Armonk, N.Y. 10504(US)

(72) Inventor: Levitan, David S.
9031 Marthas Drive
Austin, Texas 78717(US)

(74) Representative: Lettieri, Fabrizio
IBM SEMEA S.p.A.,
Direzione Brevetti,
MI SEG 024,
P.O. Box 137
I-20090 Segrate (Milano) (IT)

(54) Method and system for supporting speculative execution of instructions.

(57) A data processing system executing speculative instructions includes a memory for storing instructions at addresses, count registers (42, 44, 46) for storing an update value, a dispatch version value and a completion version value. A fetcher connected to a branch unit fetches instructions from memory based upon addresses calculated by the branch unit, which handles processing of conditional branch instructions. Further included are means (60) responsive to completion of initialization for copying the update value as the completion version value and means (52) responsive to dispatch of a conditional branch instruction. Means (62) responsive to completion of the branch provide for decrementing contents of a completion version register. Finally, means (58) responsive to occurrence of an interrupt prior to completion of the branch provide for replacing the dispatch version value with the completion version value to restore the system to a state prior to the speculative execution of instructions.

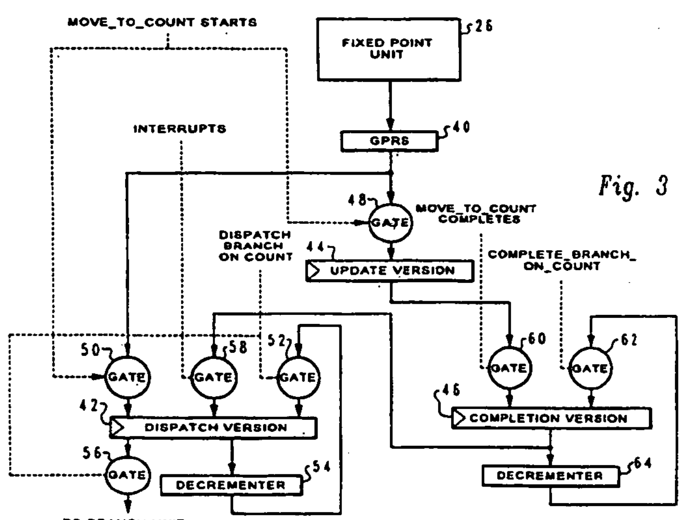

Fig. 3

EP 0 605 872 A1

The invention relates to data processing systems and in particular to a method and system for supporting speculative execution of program instructions. Still more particularly, the invention relates to preservation of non-conditional state information for recovery after speculative execution fails.

Designers of data processing systems are continually attempting to enhance the performance of such systems. One technique for enhancing data processing system efficiency is the achievement of short cycle times and a low Cycles-Per-Instruction (CPI) ratio in the system processor. An example of the application of these techniques to data processing system is the International Business Machines Corporation RISC System/6000(RS/6000)-computer. The RS/6000 system is designed to perform well in numerically intensive engineering and scientific applications as well as in multi-user, commercial environments. The RS/6000 processor employs a superscalar implementation, which means that multiple instructions are issued and executed concurrently.

Processor architecture relates to the combination of registers, arithmetic units and control logic to build the computational elements of a computer. An important consideration during building of a processor is the instruction set it will provide. An instruction is a statement which specifies an operation and the values or locations of its operands. An instruction set is the collection of all such valid statements for a particular machine.

As originally conceived, RISC machines would execute one instruction per machine cycle. To this end all instructions were of one length and fit a scheme compatible with a pipeline implementation. Simplicity in the instruction set was the design objective. This allowed further reduction in the cycle time compared with so called complex instruction set computers (CISC). However, some of the benefits of RISC were offset by increases in traffic between the processor and the main memory for a computer. This occurred because a RISC machine requires more instruction instances to do a task than a CISC machine with its more powerful instruction set.

Concurrence in issuance and execution of multiple instructions requires independent functional units that can execute with a high instruction bandwidth. The RS/6000 system achieves this by utilizing separate branch, fixed point and floating point processing units which are pipelined in nature. The branch processing unit handles conditional branch instructions. In common with other RISC designs, complex decoding logic no longer required to decode instructions has been utilized to provide an instruction cache on the processor chip. This r -duces traffic between the processor and memory, and makes fetches of instructions extrem ly fast.

An instruction subset of great interest is that relating to conditional branches. Conditional branch instructions are instructions which dictate the taking of a specified conditional branch within an application in response to a selected outcome of the processing of one or more other instructions. A practical example is a Fortran do-loop. Conditional branch instructions have long been a source of difficulty for pipeline computers (including RISC systems). By the time a conditional branch instruction propagates through a pipeline queue to an execution position within the queue, it will have been necessary to load instructions corresponding to one branch into the queue behind the conditional branch instruction prior to resolving the conditional branch, in order to avoid run-time delays. This requires a choice be made as to which instruction will follow the conditional branch without knowing the outcome of processing the related instructions. The choice can prove wrong.

The execution of instructions prior to the final possible definition of all conditions effecting execution is called speculative execution. To wait for the outcome of conditional branches, or the arrival of all possible interrupts, would make full concurrent processing impossible. Thus, some scheme for processor recovery from speculative execution of instructions must be provided if full use of concurrent execution of instructions is to be made. Upon determination that execution is proceeding down an incorrect branch an interrupt may be generated to change the course of execution. In responding to an interrupt, the processor is returned to the last non- speculative execution step.

Experience has demonstrated that use of some complex operations in RISC machines can improve performance. This in part stems from the nature of currently preferred technology for implementation of processors, i.e. very large scale integration (VLSI). Minimization of area used on a chip is now more important than minimizing the number of devices used to implement the processor. Hence, some complex instructions have begun infiltrating into RISC based designs. The criteria for inclusion is minimum utilization of space. One instruction in the RS/6000 instruction set allows execution of a branch on count loop. The branch on count instruction is a one step instruction replacing what was formerly done in three instructions. Substitution of a single instruction for three instructions was enabled by providing a dedicated count register. However, this arrangement does not in itself support speculative execution. Implementation of the count register could be done by a mechanism provided in RS/6000 machines for register rename, but the value for the count r gister would not be known during the dispatch cycle resulting in some

loss of machine cycles.

Desirable is a hardware implementation of the branch on count loop which uses a minimum amount of area on a processor chip.

It is therefore one object of the invention to provide an improved method and system for supporting speculative execution of program instructions.

It is another object of the invention to provide preservation of conditional state information for recovery after speculative execution fails.

The foregoing objects are achieved by the invention as claimed. The invention provides a data processing system for speculatively executing instructions. The data processing system includes a memory for storing instructions at addresses which can be generated by a branch unit in a processor. The processor also has a count register for storing an update value, a dispatch version value and a completion version value of a branch control count. A fetcher connected to the branch unit fetches instructions from memory based upon addresses calculated by the branch unit. The branch unit handles processing of conditional branch instructions. To do so, means for initializing the update value and the dispatch version value for branch control are provided. Further included are means responsive to completion of initialization for copying the update value as the completion version value. The system further includes means responsive to dispatch of a conditional branch instruction for examining the dispatch version value to determine if a branch should be taken and then decrementing the dispatch version value. Means responsive to completion of the branch provide for decrementing contents of a completion version register. Finally, means responsive to occurrence of an interrupt prior to completion of the branch provide for replacing the dispatch version value with the completion version value to restore the system to a state prior to the speculative execution of instructions.

The novel features believed characteristic of the invention are set forth in the appended claims. The invention itself however, as well as a preferred mode of use, further objects and advantages thereof, will best be understood by reference to the following detailed description of an illustrative embodiment when read in conjunction with the accompanying drawings, wherein:

Figure 1 is a high level block diagram of a superscalar computer system which may be utilized to implement the method and system of the present invention;

Figure 2 is a reservation table illustrating the manipulation of instruction queue content in a prior art data processing system utilizing an instruction queue; and

Figure 3 is a schematic illustration of a branch on count register architecture in accordance with a preferred embodiment of the invention; and

Figure 4 is a schematic illustration of a branch on count register architecture in accordance with a second preferred embodiment of the invention.

With reference now to the figures and in particular with reference to Figure 1, there is depicted a high level block diagram of a superscalar computer system 10 which may be utilized to implement the method and system of the present invention. As illustrated, computer system to preferably includes a memory 18 which is utilized to store data, instructions and the like. Data or instructions stored within memory 18 are preferably accessed utilizing cache/memory interface 20 in a method well known to those having skill in the art. The sizing and utilization of cache memory systems is a well known subspecialty within the data processing art is not addressed within the present application. However, those skilled in the art will appreciate that by utilizing modern associative cache techniques a large percentage of memory accesses may be achieved utilizing data temporarily stored within cache/memory interface 20.

Instructions from cache/memory interface 20 are typically loaded into instruction queue 22 which preferably includes a plurality of queue positions. In a typical embodiment of a superscalar computer system the instruction queue may include eight queue positions and thus, in a given cycle, between zero and eight instructions may be loaded into instruction queue 22, depending upon how many valid instructions are passed by cache/memory interface 20 and how much space is available within instruction queue 22.

As is typical in such superscalar computer systems, instruction queue 22 is utilized to dispatch instructions to multiple execution units. As depicted within Figure 1, computer system 10 includes a floating point processor unit 24, a fixed point processor unit 26, and a branch processor unit 28. Thus, instruction queue 22 may dispatch between zero and three instructions during a single cycle, one to each execution unit.

In addition to sequential instructions dispatched from instruction queue 22, so-called "conditional branch instructions" may be loaded into instruction queue 22 for execution by the branch processor. A conditional branch instruction is an instruction which specifies an associated conditional branch to be taken within the application in response to a selected outcome of processing one or more sequential instructions. In an effort to minimize runtime delay in a pipelined processor system, such as computer system 10, the presence of a conditional branch instruction within the instruction queue is detected and an outcome of the con-

ditional branch is predicted. As should be apparent to those having skill in the art when a conditional branch is predicted as "not taken" the sequential instructions within the instruction queue simply continue along a current path and no instructions are altered. However, if the prediction as to the occurrence of the branch is incorrect, the instruction queue must be purged of sequential instructions, which follow the conditional branch instructions in program order and target instructions must be fetched. Alternately, if the conditional branch is predicted as "taken" then the target instructions are fetched and utilized to follow the conditional branch, if the prediction is resolved as correct. And of course, if the prediction of "taken" is incorrect the target instructions must be purged and the sequential instructions which follow the conditional branch instruction in program order must be retrieved.

As illustrated, computer system 10 also preferably includes a condition registers 32. Condition registers 32 are utilized to temporarily store the results of various comparisons which may occur utilizing the outcome of sequential instructions which are processed within computer system 10. Thus, floating point processor unit 24 and fixed point processor unit 26 are coupled to condition registers 32. The status of a particular condition within condition registers 32 may be detected and coupled to branch processor unit 28 in order to generate target addresses, which are then utilized to fetch target instructions in response to the occurrence of a condition which initiates a branch.

Thereafter, branch processor unit 28 couples target addresses to fetcher 30. Fetcher 30 calculates fetch addresses for the target instructions necessary to follow the conditional branch and couples those fetch addresses to cache/memory interface 20. As will be appreciated by those having skill in the art, if the target instructions associated with those fetch addresses are present within cache/memory interface 20, those target instructions are loaded into instruction queue 22. Alternately, the target instructions may be fetched from memory 18 and thereafter loaded into instruction queue 22 from cache/memory interface 20 after a delay required to fetch those target instructions.

The manipulation of instruction queue content in a prior art data processing system utilizing an alternate instruction queue is illustrated in Figure 2 within reservation table 36 therein. Figure 2 depicts a table illustrating manipulation of instruction queue data content through seven consecutive cycle times. It may be seen that at cycle time 1, the instruction queue includes a conditional branch instruction (bc), a compare instruction (cmp) and four arithmetic logic unit (alu) instructions. Upon the detection of the conditional branch instruction within queue 3 of th prior art instruction queue, the sequential instructions within the queue are loaded into an alternate instruction queue (not shown). Thereafter, a request for target instructions associated with the conditional branch is initiated at cycle 2 and those instructions are loaded into the instruction queue at cycle 3. These instructions are based upon the prediction that the conditional branch associated with the conditional branch instruction will be "taken."

Thereafter, at cycle 4, the compare (cmp) instruction has propagated to the execution position within the instruction queue and the conditional branch instruction is "resolved." In the event the resolution of the conditional branch instruction indicates that the conditional branch is "not taken" the sequential instructions previously loaded into the alternate instruction queue are once again loaded into the primary instruction queue, as depicted at cycle 5. Cycles 6 and 7 within the instruction queue of Figure 2 indicate the subsequent processing of additional sequential instructions. As illustrated, only a single empty cycle is present within the instruction queue following the misprediction of the conditional branch instruction. However, as described above, the implementation of this prior art technique requires the utilization of an alternate instruction queue.

Referring to Figure 3, a block diagram schematic of a region of registers is depicted, including general purpose register 40 and three dedicated registers used to implement a branch on count instruction with predictive branching, speculative execution of instructions and recovery from speculative execution of instructions along a wrong branch. The three dedicated registers include a dispatch version register 42, an update version register 44 and a completion version register 46.

Functionally, the dispatch version register 42 provides a dispatch stage version of the count to be used for address generation. The dispatch stage version always leads the completion stage version of the count stored in completion version register 46. The completion stage version count is the count corresponding to the last speculatively executed and confirmed instruction. The contents of update version register 44 correspond to the count for an instruction speculatively executed, which will be copied into register 46 when confirmed. The purpose of the update version of the count is to enable decrementing of the dispatch version before the MOVE_TO_COUNT instruction completes execution. The MOVE_TO_COUNT instruction may itself be speculatively xecuted.

The movement of data into registers 42, 44 and 46 is controlled by occurrence of certain instructions from the processor instruction set. The instructions are generated by a compiler and de-

coded by conventional control logic of the processor unit. Speculative execution of a branch of a loop in a program is initiated by fixed point unit 26. A store instruction loading a count value into general purpose register 40 is executed. Subsequently execution of a MOVE_TO_COUNT instruction begins, resulting in application of a gate control signal to gates 48 and 50. As a result the contents of general purpose register 40 are copied into dispatch version register 42 and update version register 44.

Upon dispatch of a BRANCH_ON_COUNT instruction the contents of register 42 are examined to determine if the branch should be taken or if execution of steps of the program should fall through sequentially. A gate control signal is applied to gates 52 and 56, which results in the application of the contents of register 42 to branch unit 28 and copying into register 42 of the prior contents of the register less 1 by route of decrementer 54.

If an interrupt is taken prior to completion of instructions following a BRANCH_ON_COUNT instruction, indicating, for example, that the wrong sequence of instruction was followed, a gate control signal is applied to gate 58. Gate 58 is used to copy the contents of completion version register 46 into dispatch version register 42. This returns the state of dispatch version register 42 to that preceding execution of any speculative instruction not yet confirmed.

As previously described, gate 48 controls the copying of the contents of general purpose register 40 to update version register 44 with initiation of a MOVE_TO_COUNT instruction. The contents of update register 44 are copied to completion version register 46 by gate 60 with completion of the MOVE_TO_COUNT instruction. There the initial count is preserved until a branch following a BRANCH_ON_COUNT instruction is completed and confirmed. Upon completion logic of the processor signal removing the tentative markings from the results of the branch, a gating signal is applied to gate 62 resulting in the contents of register 46 being decremented by decrementer 64 and the result being copied back into register 46.

Figure 4 illustrates a simplified but lower performance embodiment of the invention, in which gate 46 and update version register 44 have been eliminated. The movement of data into dispatch version register 42 and completion version register 46 is controlled by signals to various gates. The completion version register 46 now receives its data from the dispatch version register 42, rather than the update version register. The signals are similar to those discussed with reference to Figure 3, but occur at somewhat different times. The most important change in timing is that for BRANCH_ON_COUNT instructions, which cannot execute until a MOVE_TO_COUNT instruction completes and the completion version register 46 has been loaded.

With initiation of a MOVE_TO_COUNT instruction, a gate signal is applied to gate 50 copying the contents of general purpose register 40 into dispatch version register 42. When the MOVE_TO_COUNT completes a signal applied to gate 60 results in the contents of the dispatch version register 42 being applied to completion version register 46. Now a DISPATCH_ON_COUNT instruction can be executed. The handling of BRANCH_ON_COUNT instructions, interrupts and complete BRANCH_ON_COUNT is identical to the first embodiment.

Claims

1. A processor for a data processing system (10) including a branch unit (28) for operating on conditional branch instructions and calculating target addresses for use in fetching instructions from a memory device (18), said processor comprising:

   a source of an initial version of a count value related to a conditional branch;

   means (42) for storing a dispatch version of the count value;

   means (46) for storing a completion version of a count value corresponding to non-speculatively executed instructions;

   means (50) responsive to beginning execution of a move to count instruction for loading the initial version of the count value into said means (42) for storing the dispatch version;

   means responsive to completion of a move to count instruction for moving initial version of the count value to said means (46) for storing the completion version;

   means (52, 54) responsive to dispatch of a branch on count instruction for decrementing the dispatch version; and

   means (62, 64) responsive to completion of a branch for decrementing the contents of said means (46) for storing the completion version.

2. A processor as set forth in Claim 1 ,wherein said means for moving the initial version of the count values includes:

   update means (44) for storing the initial version of the count; and

   a gate (60) for transferring the content of said update means (44) to said means (46) for storing the completion version.

3. A processor as set forth in Claim 2, and further comprising:
means (58) responsive to an interrupt for loading the contents of said means (46) for storing the completion version into said means (42) for storing the dispatch version.

4. In a data processing system (10) having a branch unit (26) in a processor for processing conditional branch instructions, a method of speculatively executing instructions recovered from memory (18) based upon addresses calculated by said branch unit (28), the method including the steps of:
initializing a dispatch version register (42) with a count value for control of the conditional branch;
upon completion of initialization of said dispatch version register (42), copying the initialization data to a completion version register (46);
responsive to dispatch of a conditional branch instruction, examining contents of said dispatch version register (42) to determine a branch to be taken and then decrementing the contents of said dispatch version register (42);
responsive to completion of the taken branch, decrementing contents of said completion version register (46); and
upon occurrence of an interrupt prior to completion of the taken branch, copying the contents of said completion version register (46) to said dispatch version register (42).

5. A method as set forth in Claim 4, and further comprising executing the steps subsequent to the step of copying the contents of an update version register (44) into said completion version register (46) as a loop.

6. A method as set forth in Claim 5, wherein the step of initializing comprises:
loading a value into a general purpose register (40) of the processor; and
executing a move of the contents of said general purpose register (40) into said dispatch version register (42) and into said update version register (44).

7. A method as set forth in Claim 6, wherein the step of copying the initialization data into said completion version register (44) comprises:
copying the contents of said update version register (44) into said completion version register (46).

8. A m thod as set forth in Claim 7, and further comprising:
dispatching a branch on count instruction without regard to the number of uncompleted branch on count instructions already queued.

9. A data processing system (10) for speculatively executing instructions including:
a memory (18) for storing instructions;
a dispatch version register (42) and a completion version register (46);
means (30) for fetching instructions from said memory (18) based upon calculated addresses;
a branch unit (28) for processing conditional branch instructions;
means (40, 50) for initializing the contents of the dispatch version register (42) for branch control;
means (60) responsive to completion of initialization for copying initialization data into said completion version register (46);
means (52) responsive to dispatch of a conditional branch instruction for examining the dispatch version value to determine a branch to take and then decrementing the contents of said dispatch version register (42);
means (62) responsive to completion of a branch for decrementing contents of said completion version register (46); and
means (58) responsive to occurrence of an interrupt prior to completion of the branch for replacing the content of said dispatch version register (42) with the contents of said completion version register (46) to restore the system to a state preceding the speculative execution of instructions

10. A data processing system (10) as set forth in Claim 9, wherein said means (40, 50) for initializing the contents of the dispatch version value comprise:
means for loading a value into a general purpose register (40); and
means (50) for executing a move of the contents of the general purpose register (40) into said dispatch version registers (42).

11. A data processing system as set forth in Claim 10, wherein the dispatch of branch on count instructions may be made without regard to the number of uncompleted branch on count instructions already queued.

12. A data processing system as set forth in Claim 11, and further comprising:
an update version register (44);
means (46) for initializing the contents of the update version (44) register synchronously with initialization of said dispatch version regis-

ter (42); and

the means (60) for copying initialization data into the completion version is connected between said update register (44) and said completion version register (46) for copying the contents of the update version register (44) into said completion version register (46).

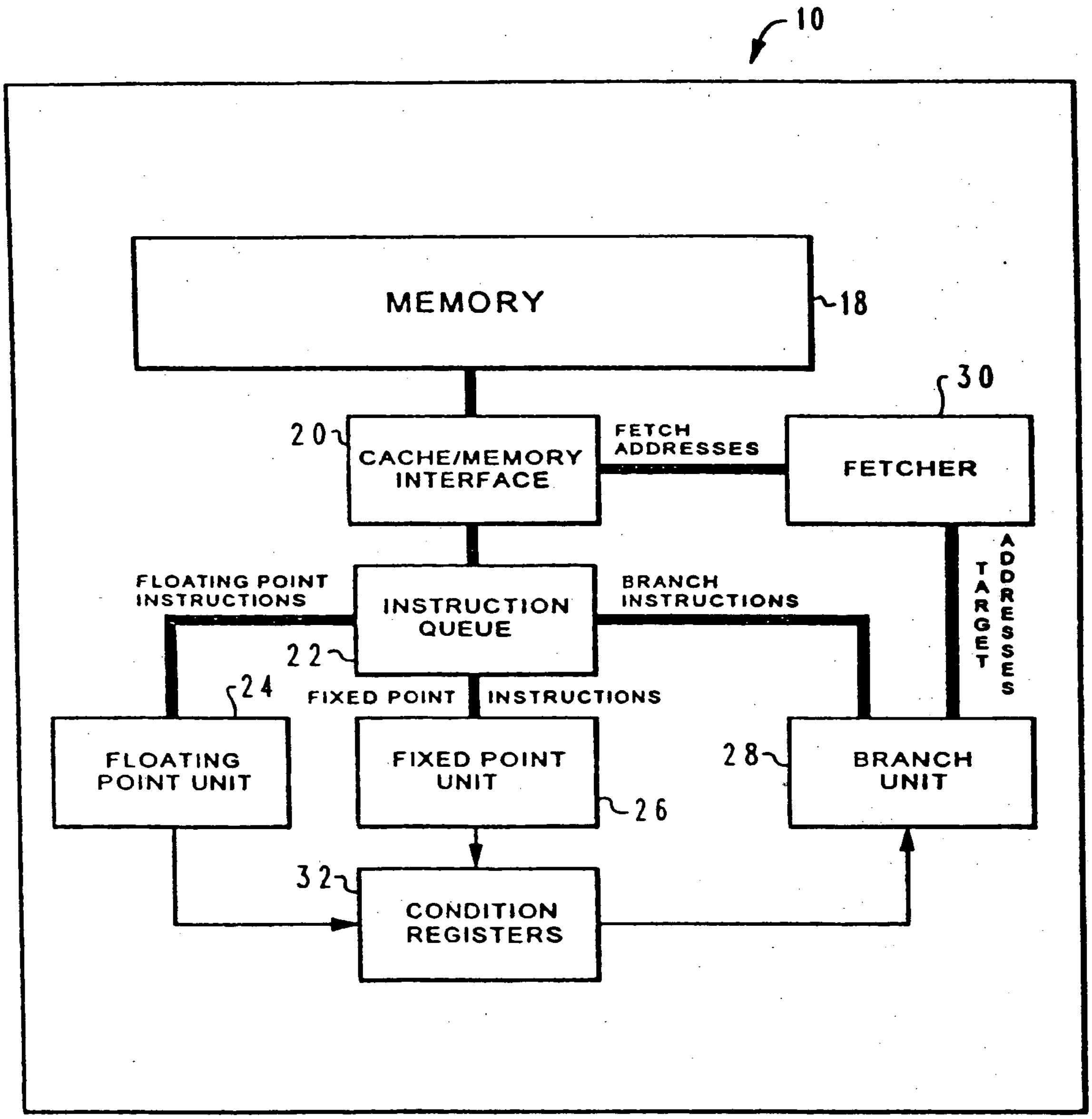
10
MEMORY
18
30
20
CACHE/MEMORY INTERFACE
FETCH ADDRESSES
FETCHER
FLOATING POINT INSTRUCTIONS
INSTRUCTION QUEUE
BRANCH INSTRUCTIONS
TARGET ADDRESSES
22
24
FIXED POINT
INSTRUCTIONS
FLOATING POINT UNIT
FIXED POINT UNIT
28
BRANCH UNIT
26
32
CONDITION REGISTERS

Fig. 1

36

Stage	Cycle 1	2	3	4	5	6	7
Queue 3	bc		T2	T3	S3	S4	S5
Queue 2	cmp		T1	T2	S2	S3	S4
Queue 1	alu	cmp	T0	T1	S1	S2	S3
Decode	alu	alu	cmp	T0	S0	S1	S2
Execute	alu	alu	alu	cmp		S0	S1
W / B	alu	alu	alu	alu	cmp		S0

Fig. 2 Prior Art

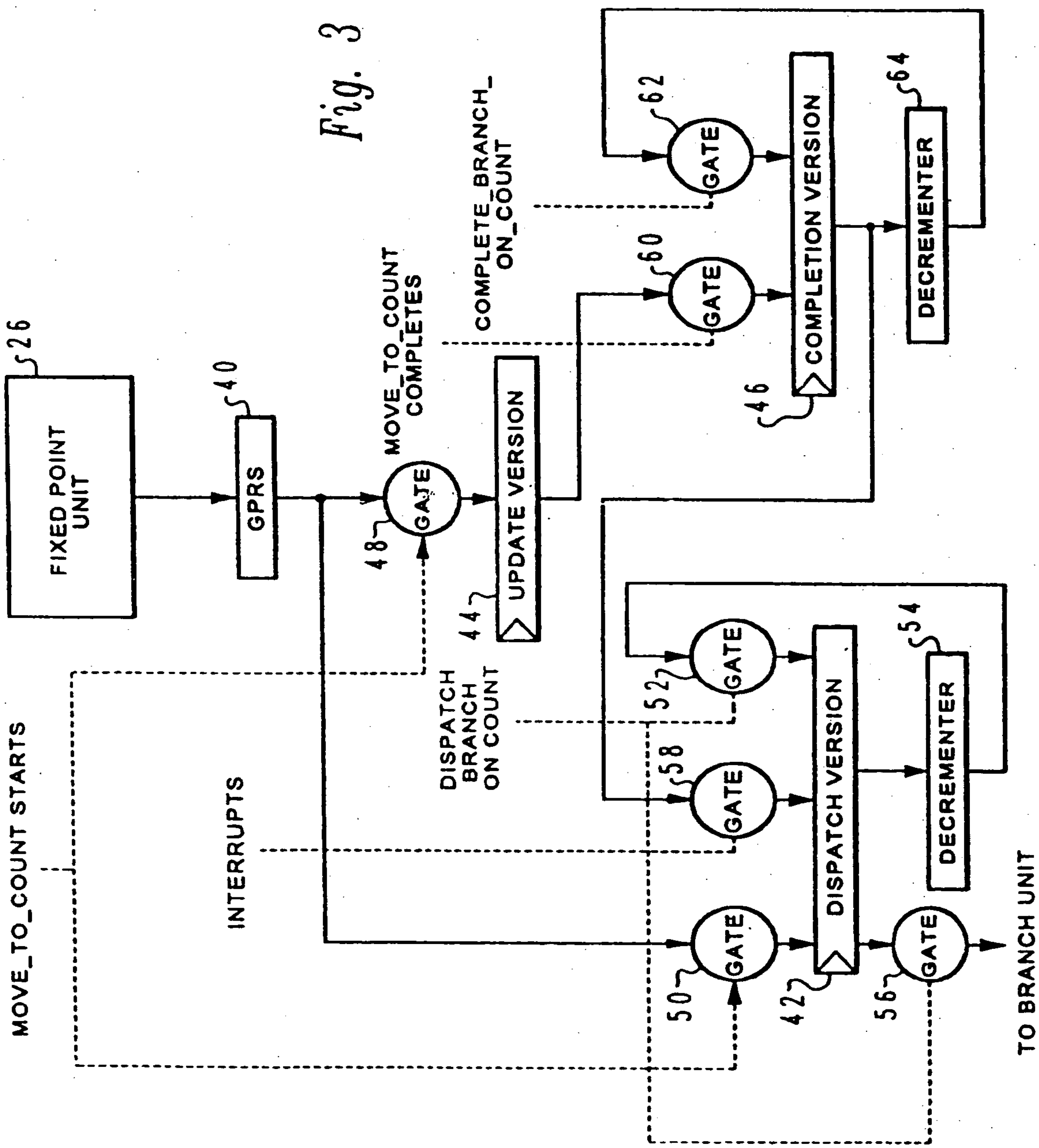
FIXED POINT UNIT
26
GPRS
40
GATE
48
MOVE_TO_COUNT COMPLETES
UPDATE VERSION
44
GATE
60
COMPLETE_BRANCH_ ON_COUNT
GATE
62
COMPLETION VERSION
46
DECREMENTER
64
MOVE_TO_COUNT STARTS
INTERRUPTS
DISPATCH BRANCH ON COUNT
GATE
52
GATE
58
GATE
50
DISPATCH VERSION
42
DECREMENTER
54
GATE
56
TO BRANCH UNIT

Fig. 3

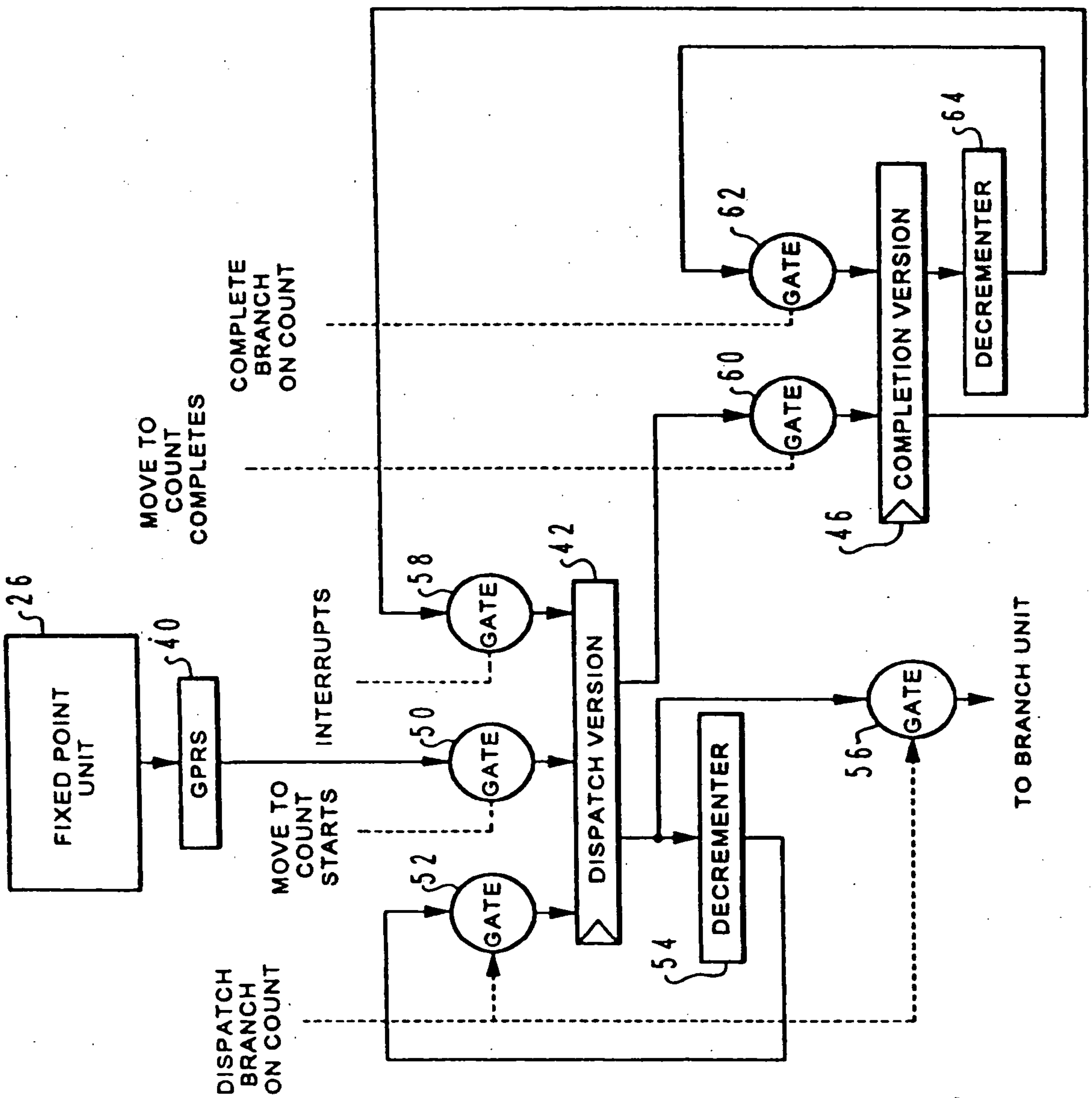
FIXED POINT UNIT
26
GPRS
40
MOVE TO COUNT STARTS
INTERRUPTS
DISPATCH BRANCH ON COUNT
MOVE TO COUNT COMPLETES
COMPLETE BRANCH ON COUNT
GATE
52
GATE
50
GATE
58
DISPATCH VERSION
42
DECREMENTER
54
GATE
56
TO BRANCH UNIT
GATE
60
GATE
62
COMPLETION VERSION
46
DECREMENTER
64

Fig. 4

European Patent Office

EUROPEAN SEARCH REPORT

Application Number
EP 93 12 0940

DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int.Cl.5)
A	US-A-5 101 484 (KOHN) * the whole document * ---	1,4,9	G06F9/38
A	IEEE TRANSACTIONS ON COMPUTERS vol. 37, no. 5 , May 1988 , NEW YORK US pages 562 - 573 SMITH AND PLESZKUN 'Implementing precise interrupts in pipelined processors' * page 566, section IV; page 568, section VI * ---	1,4,9	
A	IBM TECHNICAL DISCLOSURE BULLETIN. vol. 36, no. 1 , January 1993 , NEW YORK US pages 262 - 264 'Looping in MSIS' * the whole document * ---	1,4,9	
A	SUPERCOMPUTING '90 12 November 1990 , NEW YORK, US pages 200 - 212 TIRUMALAI, LEE AND SCHLANSKER 'Parallelization of loops with exits on pipelined architectures' * page 201, right column, line 35 - page 202, left column, line 16 * -----	1,4,9	TECHNICAL FIELDS SEARCHED (Int.Cl.5) G06F

The present search report has been drawn up for all claims

Place of search	Date of completion of the search	Examiner
THE HAGUE	25 April 1994	Weinberg, L

CATEGORY OF CITED DOCUMENTS

X : particularly relevant if taken alone
Y : particularly relevant if combined with another document of the same category
A : technological background
O : non-written disclosure
P : intermediate document

T : theory or principle underlying the invention
E : earlier patent document, but published on, or after the filing date
D : document cited in the application
L : document cited for other reasons

& : member of the same patent family, corresponding document